Estruturas de Dados - ESP412

Profª Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

March 27, 2025

Árvores Binárias

Árvores Binárias

Árvores Binárias

Definição:

Árvores Binárias são Estruturas de Dados caracterizadas por possuir nós chamados "pais" que podem possuir zero, um ou dois filhos, por isso chama-se ÁRVORE BINÁRIA, pois cada nó pai pode possuir, no máximo, dois filhos.

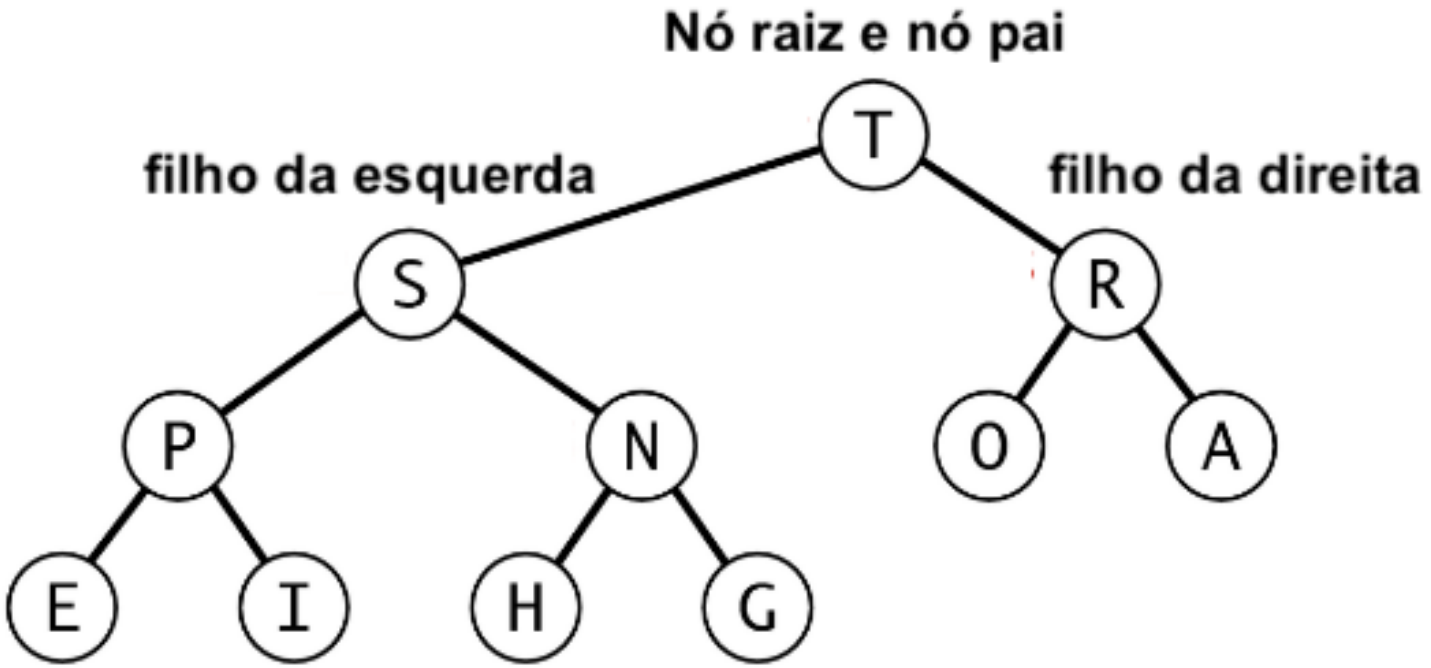
Árvores Binárias

Definição:

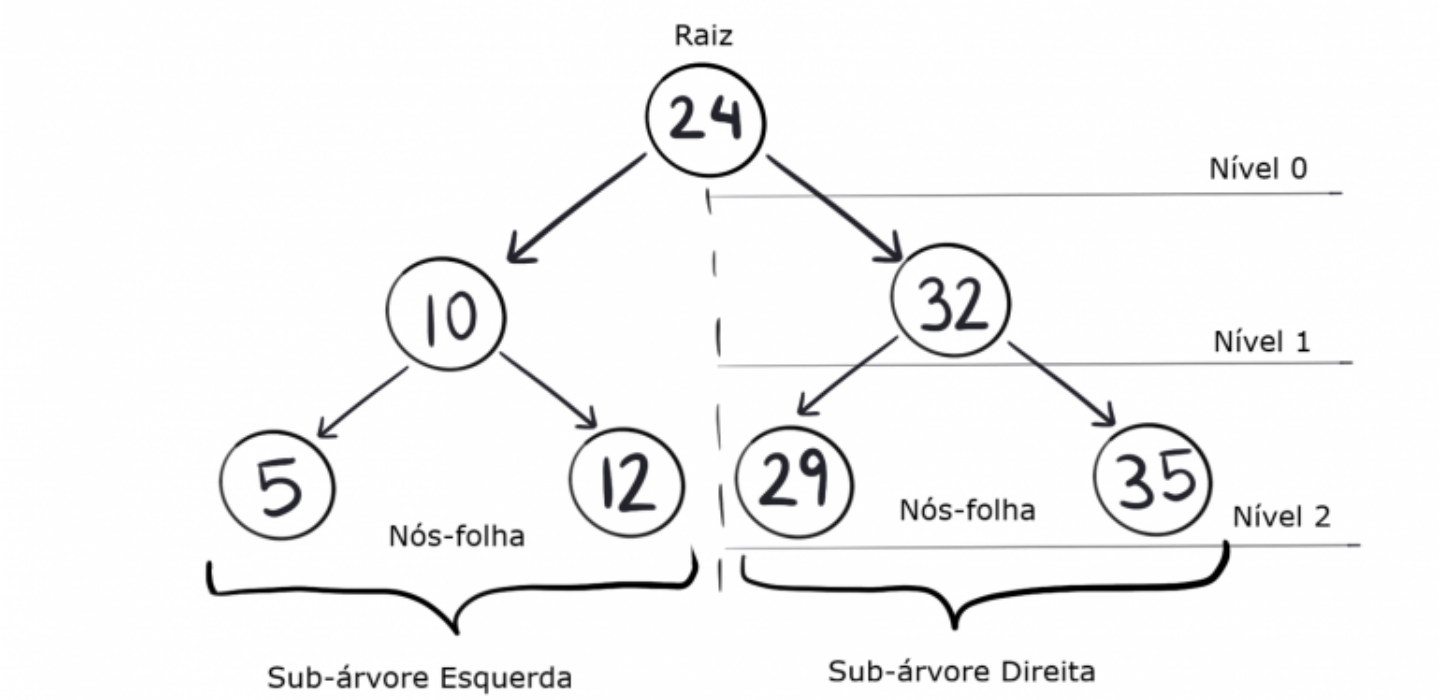
Árvores Binárias são Estruturas de Dados caracterizadas por possuir nós chamados "pais" que podem possuir zero, um ou dois filhos, por isso chama-se ÁRVORE BINÁRIA, pois cada nó pai pode possuir, no máximo, dois filhos.

Cada nó possui um identificador ID, e dois ponteiros para seus filhos da esquerda e direita (LEFT e RIGHT). E o nó pai pode ser filho de alguém ou pode ser a "RAIZ" da Árvore.
[Cormen et al. 2009]

Árvores Binárias



Árvores Binárias



Árvores Binárias

```
typedef struct noArvore{
   int chave;
   struct noArvore *filhoDir;
   struct noArvore *filhoEsq;
}NO;
```

Árvores Binárias

Outra definição:

Uma árvore binária é uma estrutura de dados caracterizada por:

Árvores Binárias

Outra definição:

Uma árvore binária é uma estrutura de dados caracterizada por:

- Ou não tem elemento algum (árvore vazia).
- Ou tem um elemento distinto, denominado raiz, com dois ponteiros para duas estruturas diferentes, denominadas subárvore esquerda e subárvore direita.

Árvores Binárias

Altura da Árvore Binária:

Nas Árvores Binárias só existe um caminho a percorrer, que é o caminho da raiz aos nós-folha. Este caminho é denominado de **Altura da árvore**, que se define pelo **maior caminho a partir de um determinado nó da árvore**.

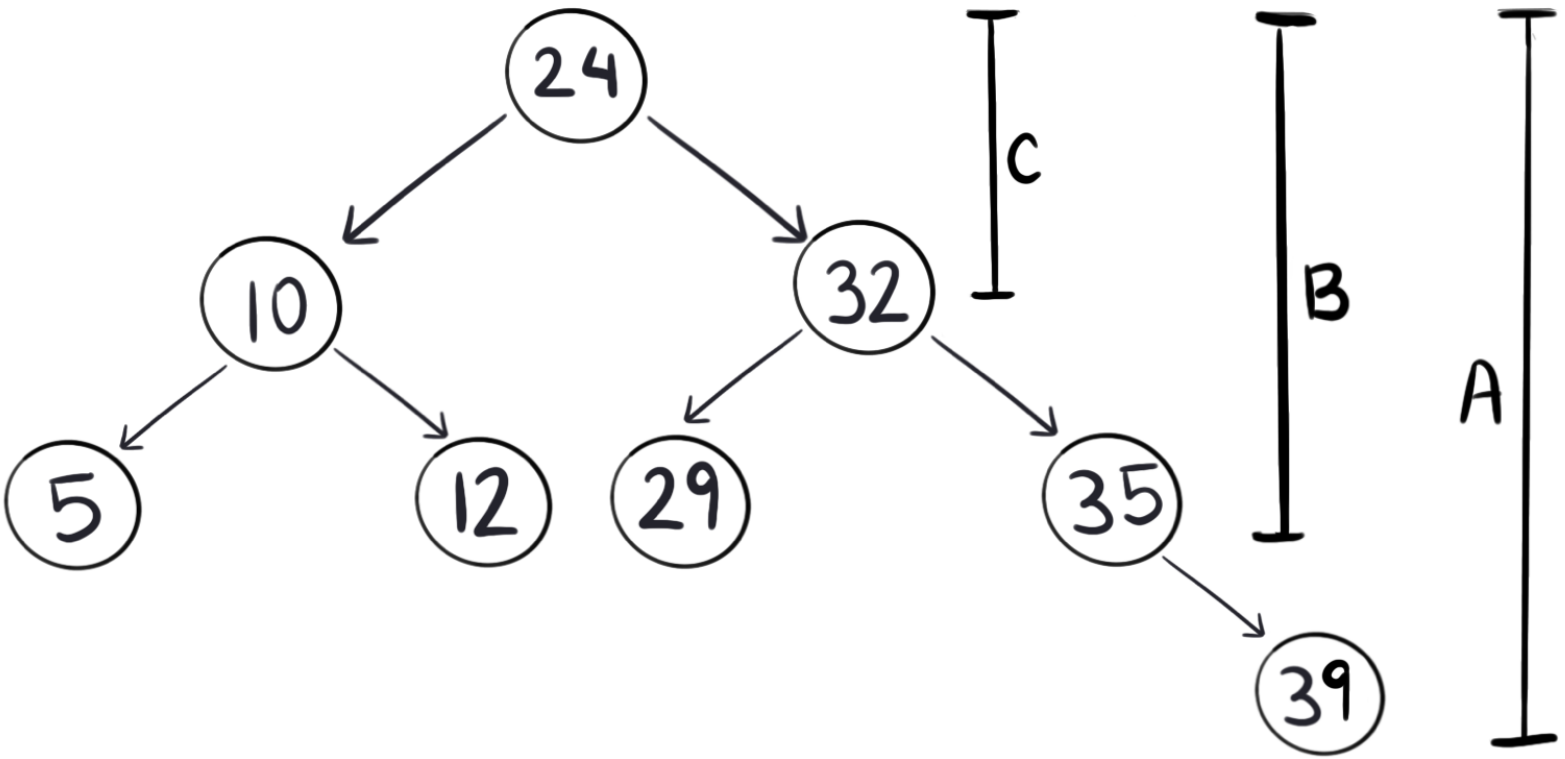
Árvores Binárias

Altura da Árvore Binária:

Nas Árvores Binárias só existe um caminho a percorrer, que é o caminho da raiz aos nós-folha. Este caminho é denominado de **Altura da árvore**, que se define pelo **maior caminho a partir de um determinado nó da árvore**.

Este também é o maior tempo de busca de um elemento contido na Árvore Binária. Por isso, as árvores são as melhores estruturas de dados para armazenamento de grandes volumes de dados em Bancos de Dados, pois o seu tempo de busca de dados é pequeno.

Árvores Binárias



Árvores Binárias

Podemos observar que:

- A é a Altura do nó raiz (24) até o nó-folha (39), então a altura de A é 3;
- B é a Altura do nó raiz (24) até o nó 35, então a altura de B é 2;
- C é a Altura do nó raiz (24) até o nó-filho 32, então a altura de C é 1.

Árvores Binárias

Podemos observar que:

- A é a Altura do nó raiz (24) até o nó-folha (39), então a altura de A é 3;
- B é a Altura do nó raiz (24) até o nó 35, então a altura de B é 2;
- C é a Altura do nó raiz (24) até o nó-filho 32, então a altura de C é 1.

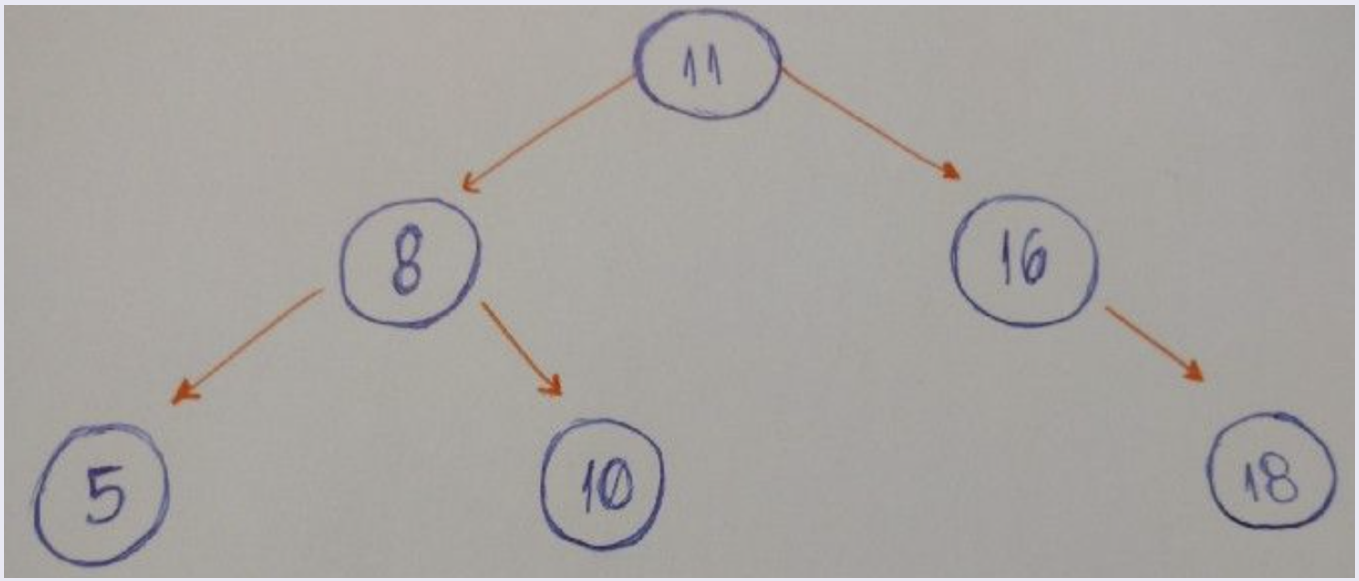
Atenção!

A Altura e o Nível de uma Árvore Binária não são a mesma coisa. Altura é a distância entre um nó e outro, ou seja, quantos nós existem entre um nó e outro. No exemplo, a Altura de A é 3 porque entre 24 e 39 existem 3 nós (o 32, o 35 e o 39).

Árvores Binárias

Árvores Binárias Ordenadas ou Árvores Binárias de Busca:

São Árvores Binárias onde os filhos de cada nó estão ordenados, assumindo ordenação da esquerda pra direita.



Árvores Binárias

Árvores Binárias Ordenadas ou Árvores Binárias de Busca:

Árvores Binárias Ordenadas são chamadas de Árvores Binárias de Busca porque sua ordenação possibilita a realização de buscas mais eficientes.

As chaves buscadas podem ser comparadas com as chaves dos nós da árvore, possibilitando decidir se devemos continuar a busca somente na subárvore à esquerda ou à direita do nó, reduzindo drasticamente a quantidade de nós a serem visitados na busca.

Inserção Ordenada

Árvores Binárias

Inserção na Árvore:

A inserção em uma Árvore Binária deve ser realizada de forma ordenada. Visando manter a árvore sempre organizada:

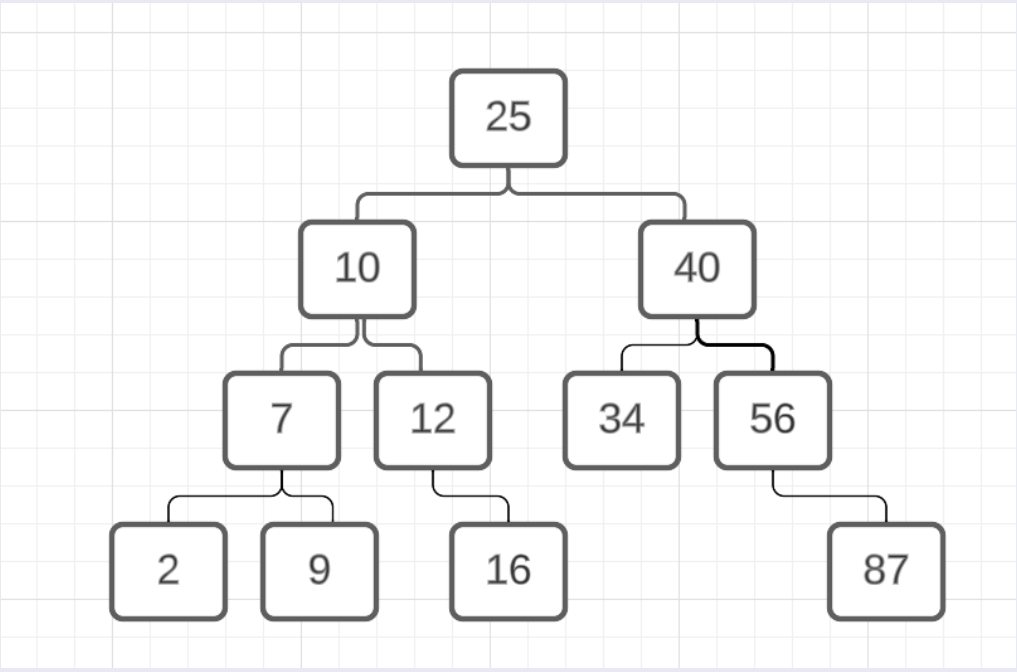
```
NO *insereNaArvore(NO *nodo, int chave){
  if (nodo == NULL){
      nodo = malloc (sizeof(NO));
      nodo->filhoEsq = NULL;
      nodo->filhoDir = NULL;
      nodo->chave = chave;
  }else{
    if(nodo->chave > chave)
      nodo->filhoEsq = insereNaArvore(nodo->filhoEsq, chave);
    else
      nodo->filhoDir = insereNaArvore(nodo->filhoDir, chave);
  }
  return nodo;
}
```

Remoção

Árvores Binárias

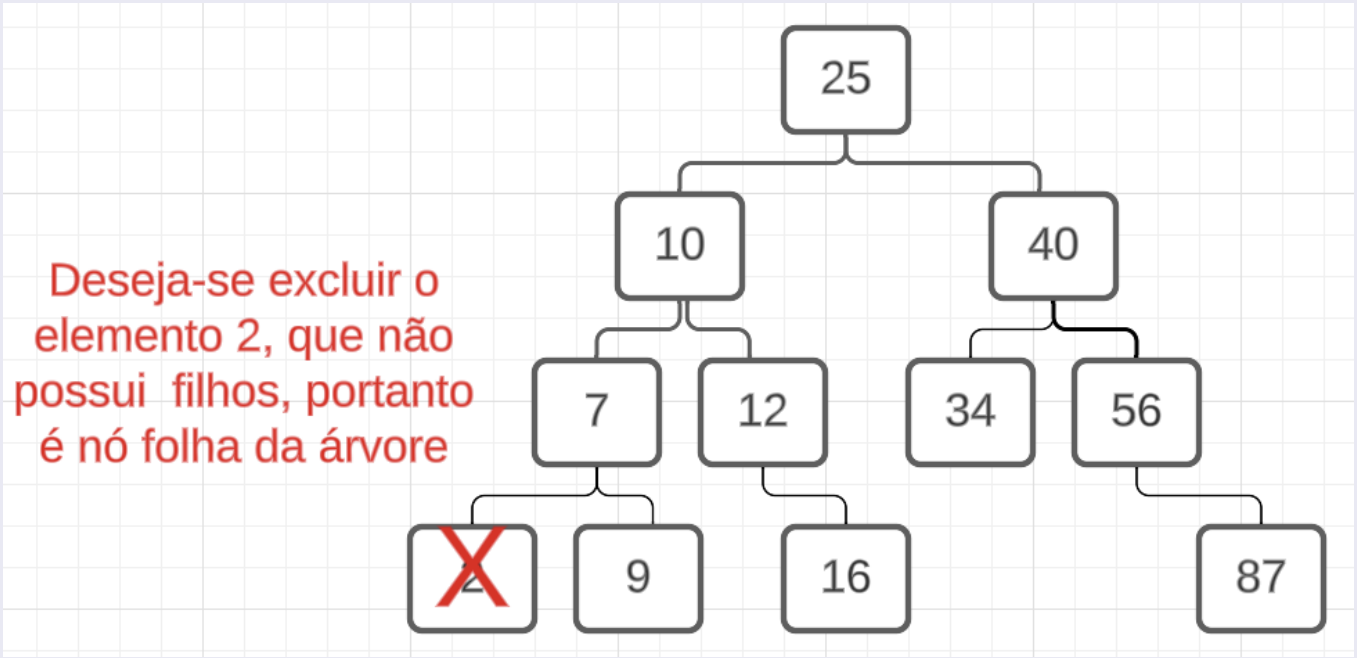
Remoção na Árvore Binária Ordenada:

A Remoção em uma Árvore Binária deve ser realizada mantendo a organização da árvore, portanto, deve-se avaliar a posição do nó a ser removido.



Árvores Binárias

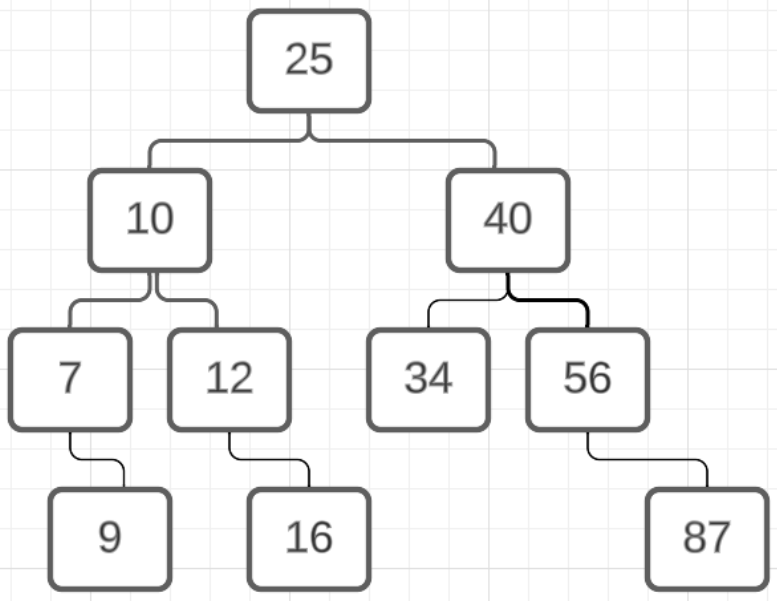
Removendo Nó Folha:



Árvores Binárias

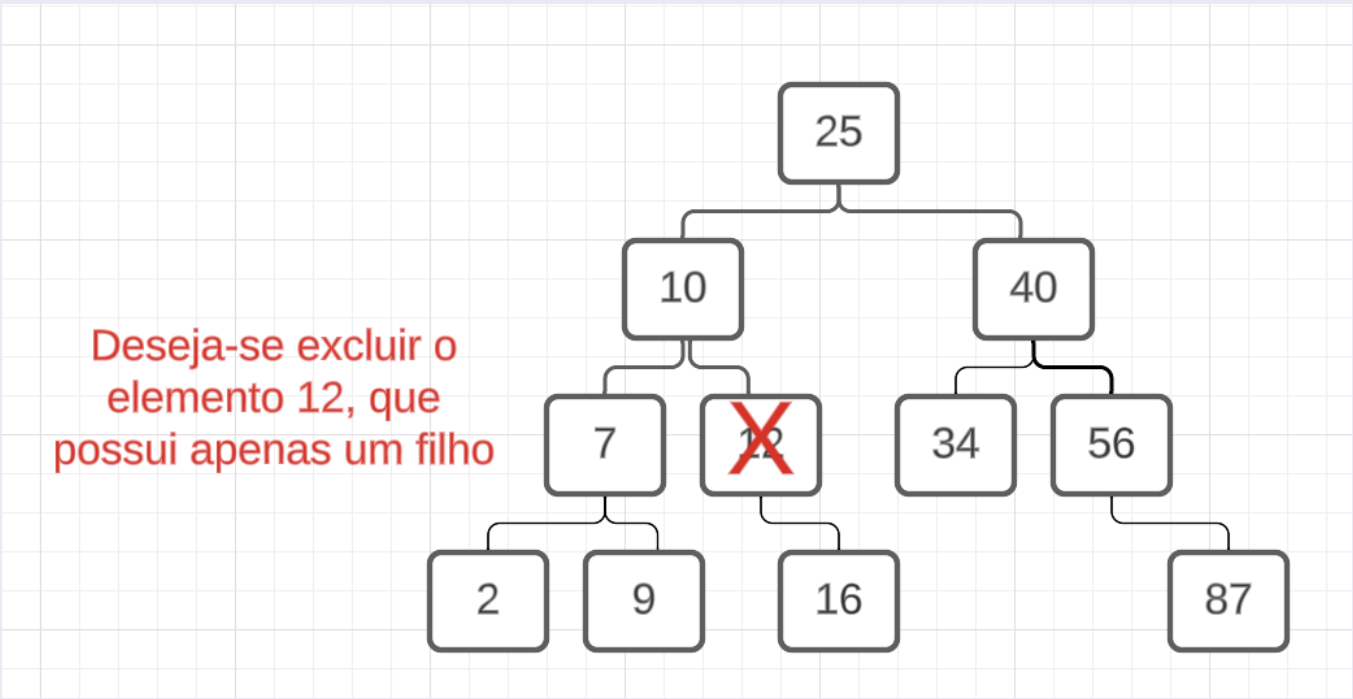
Removendo Nó Folha:

Basta apagar o nó e setar o filho esquerdo do pai, nesse caso o nó 7, para nulo



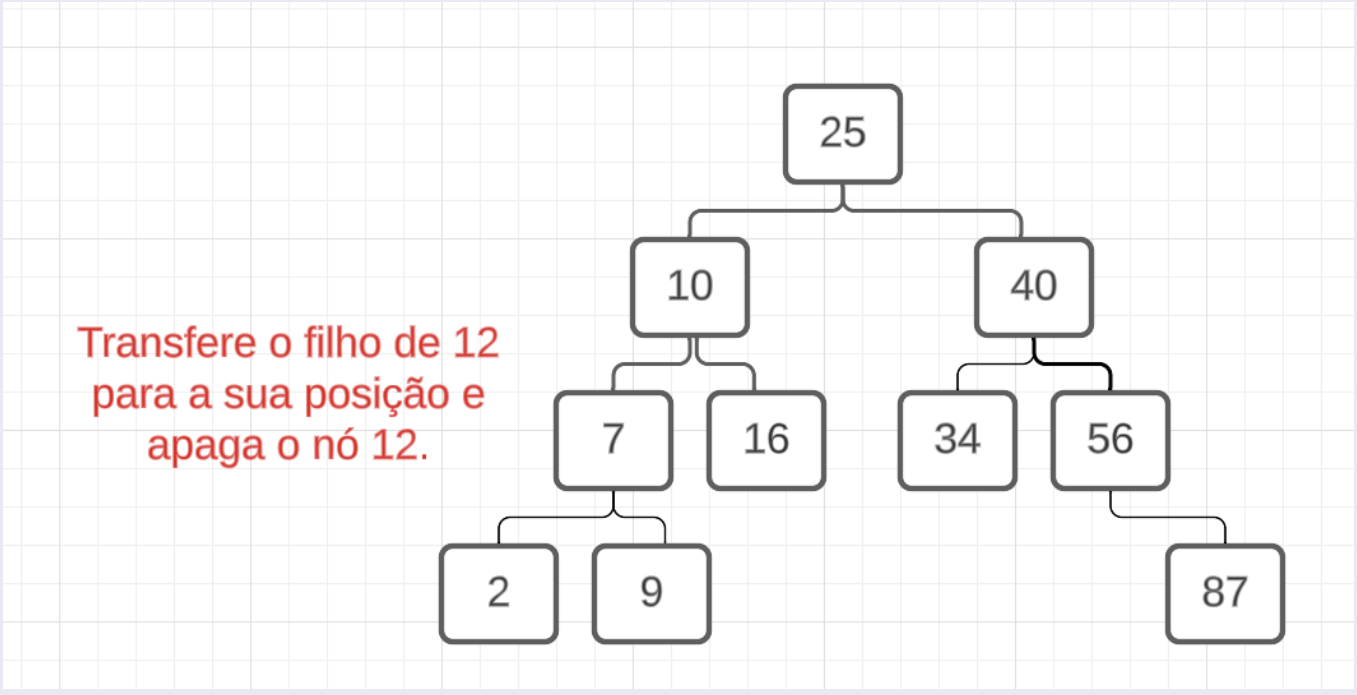
Árvores Binárias

Removendo um nó que possui UM filho, seja filho à esquerda ou à direita:



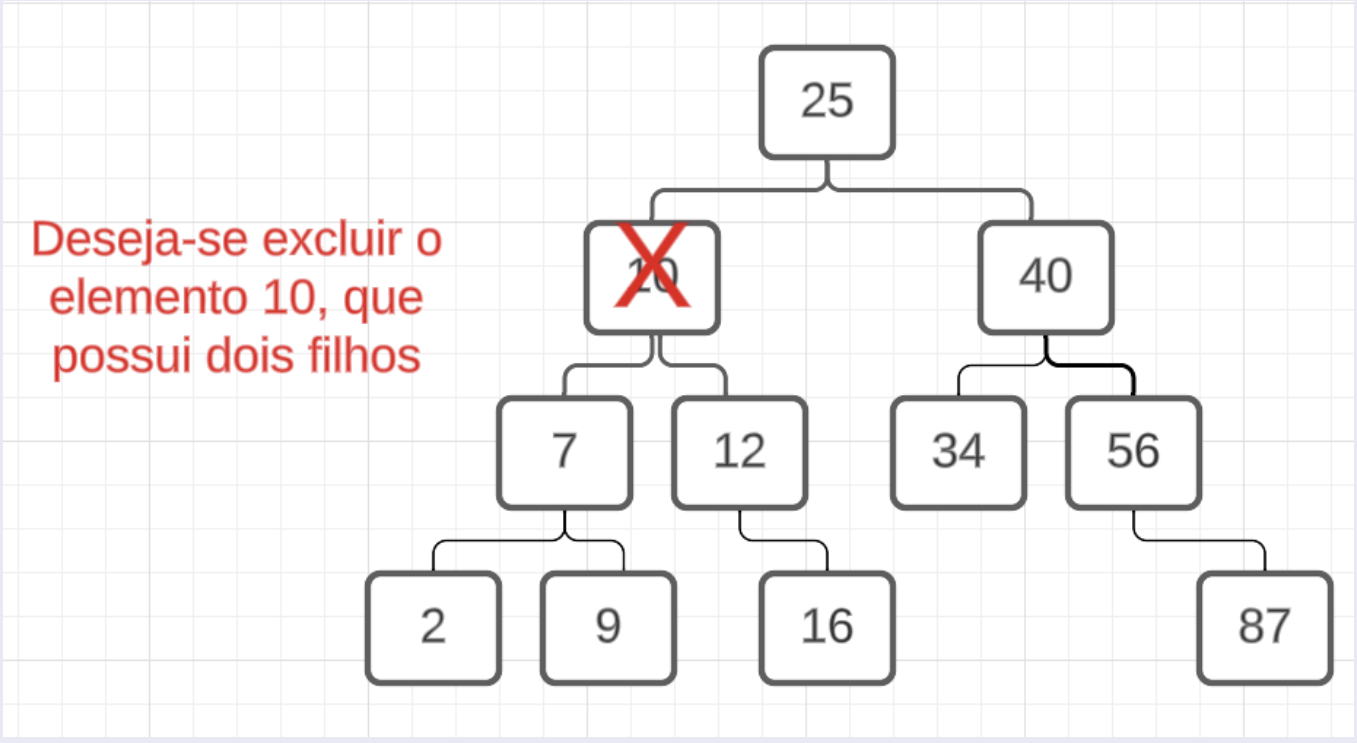
Árvores Binárias

Removendo um nó que possui UM filho, seja filho à esquerda ou à direita:



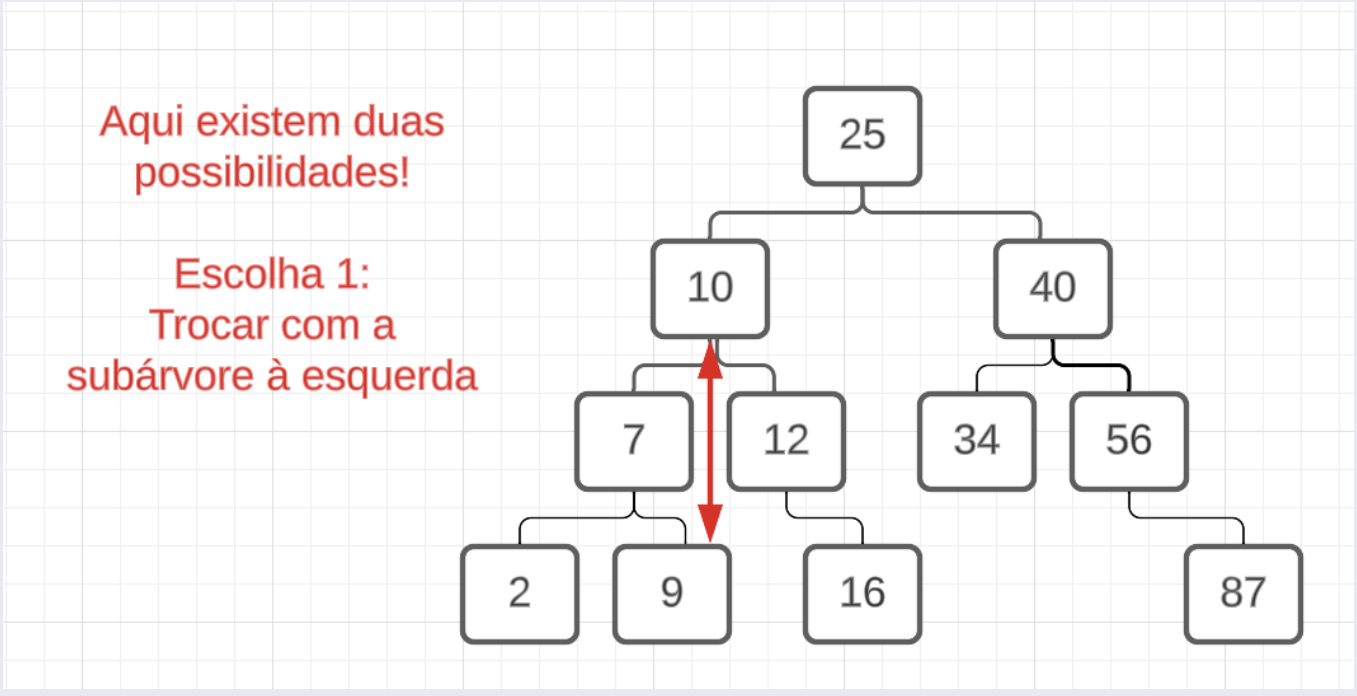
Árvores Binárias

Removendo um nó que possui DOIS filhos:



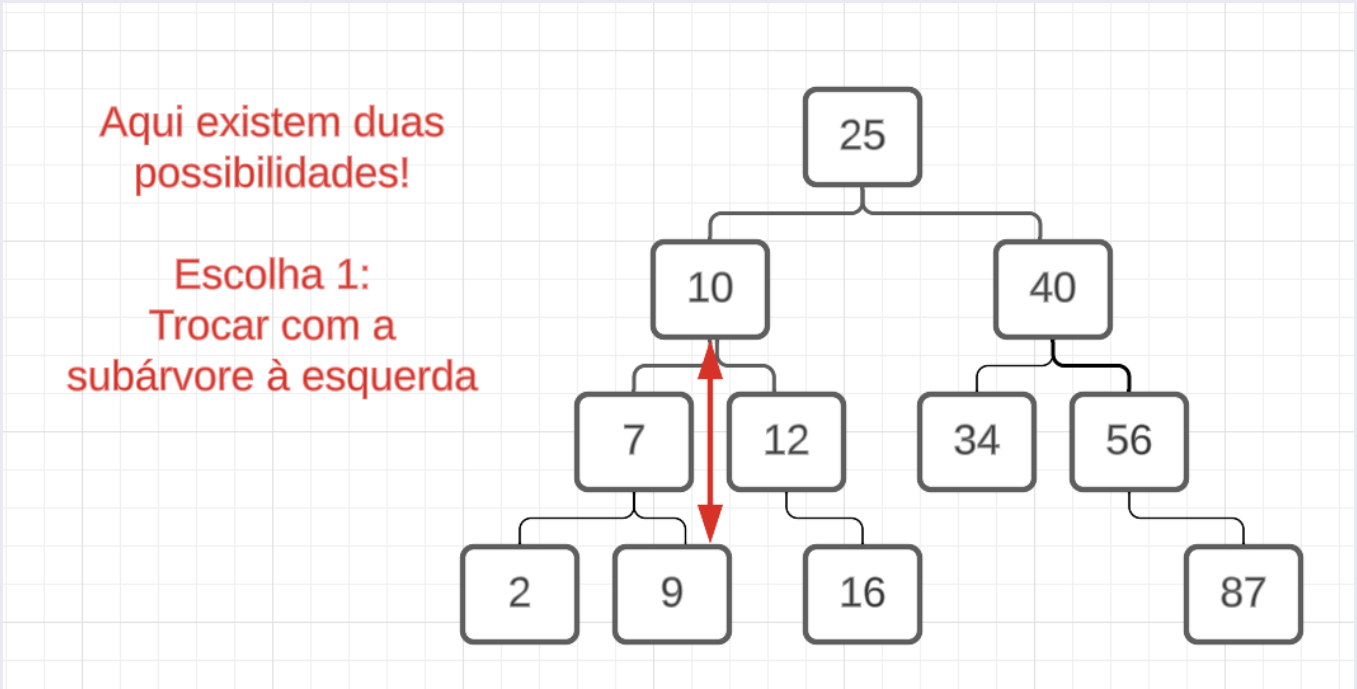
Árvores Binárias

Removendo um nó que possui DOIS filhos:



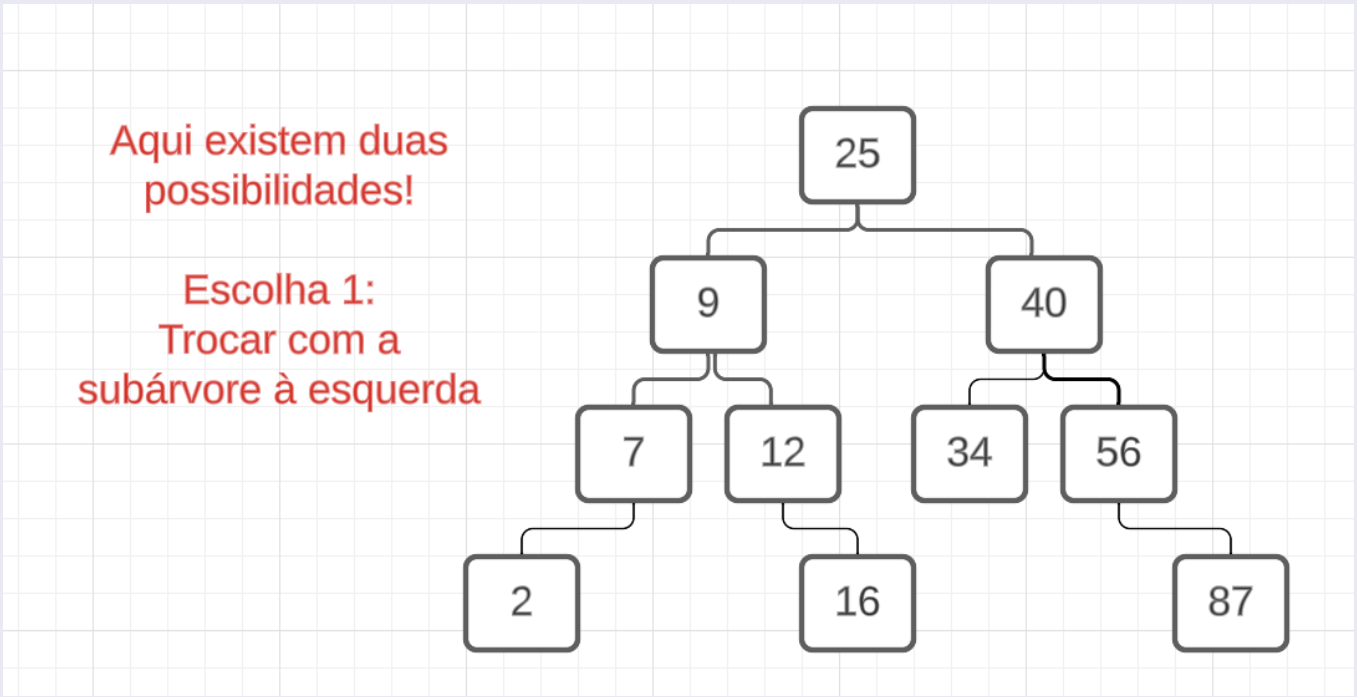
Árvores Binárias

Removendo um nó que possui DOIS filhos: Ao escolher trocar com a subárvore à esquerda, deve-se pegar o maior elemento à esquerda. Ele será o substituto do elemento excluído. Este elemento é o elemento mais à direita da subárvore à esquerda.



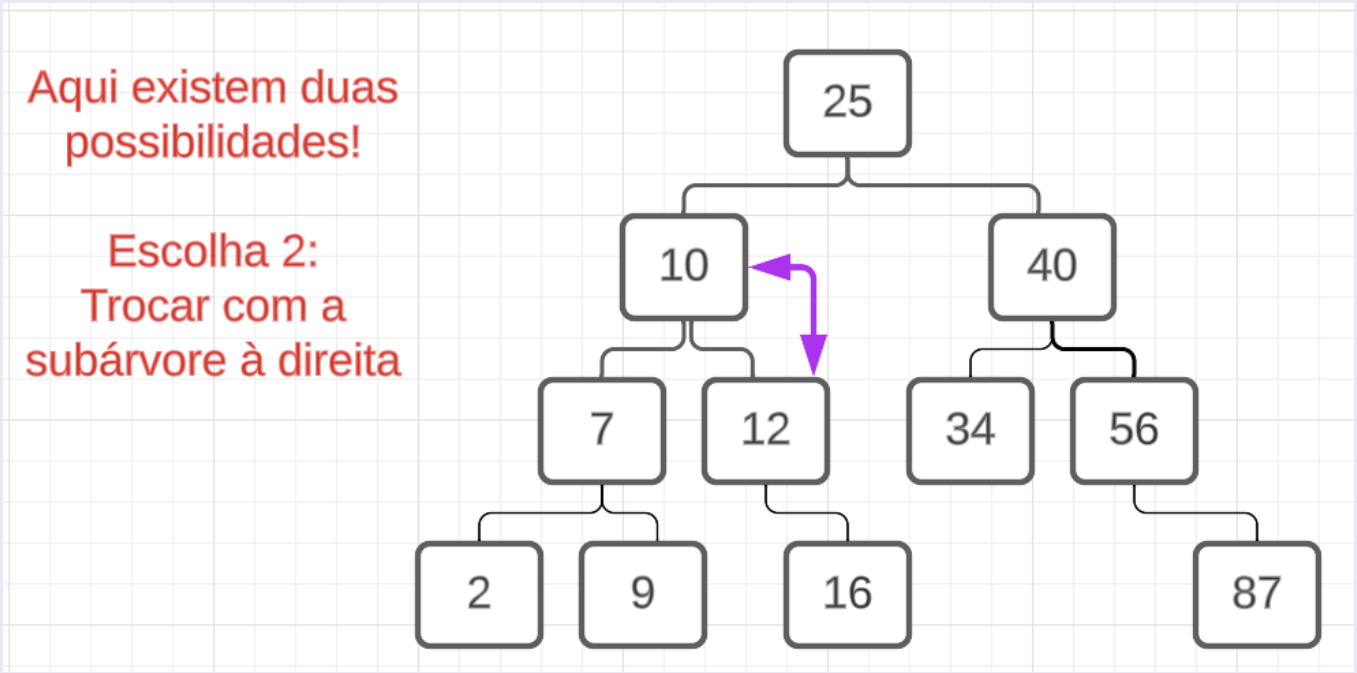
Árvores Binárias

Removendo um nó que possui DOIS filhos:



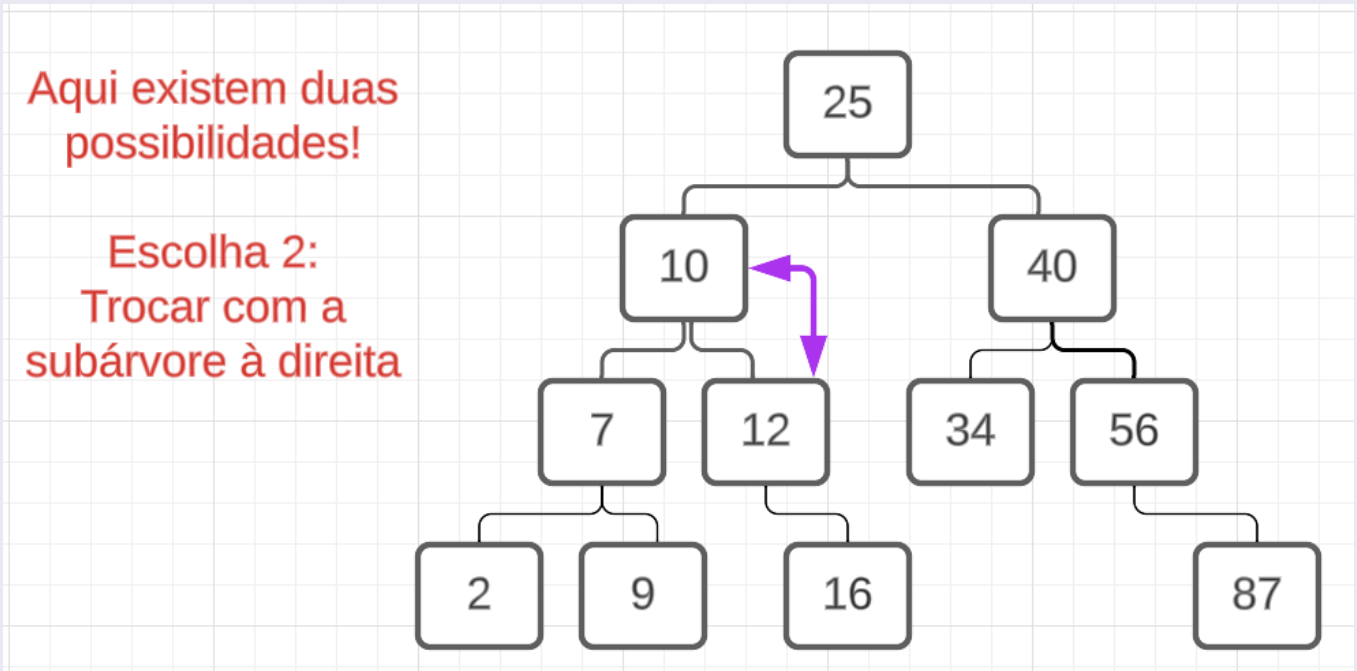
Árvores Binárias

Removendo um nó que possui DOIS filhos:



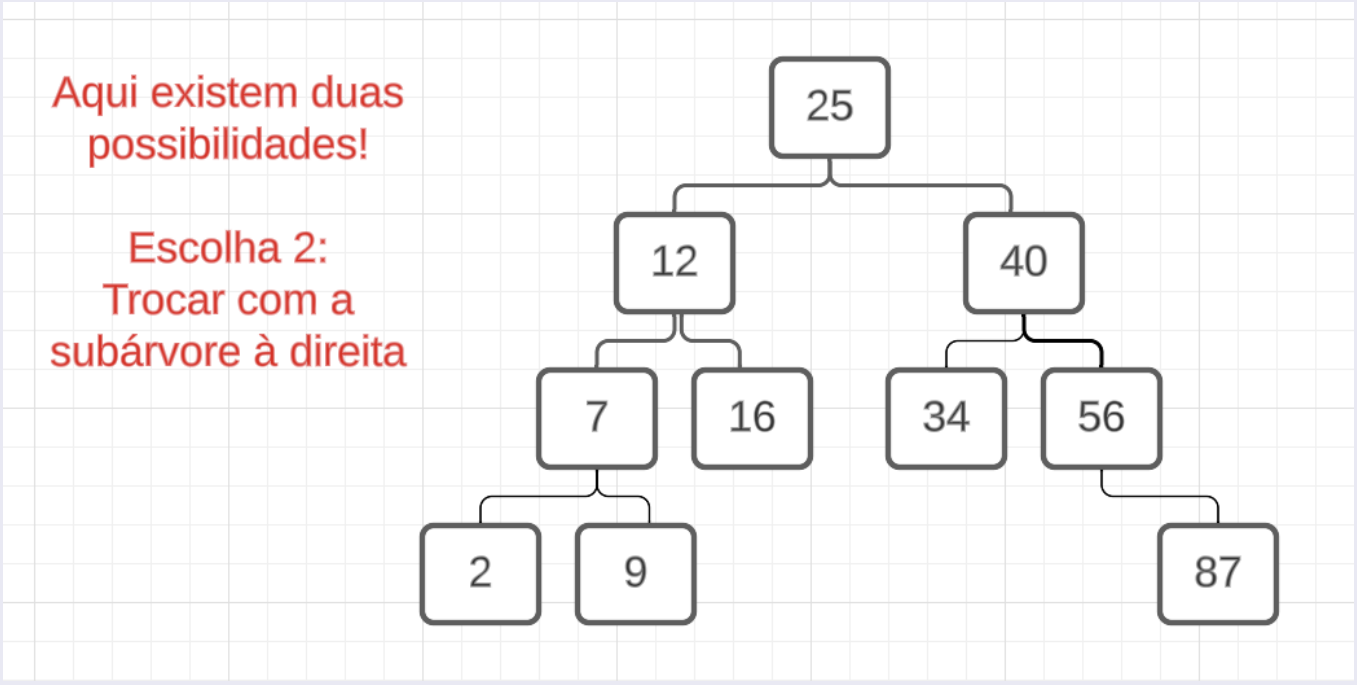
Árvores Binárias

Removendo um nó que possui DOIS filhos: Analogamente, Ao escolhermos trocar com a subárvore à direita, selecionamos o elemento mais à esquerda da subárvore à direita.



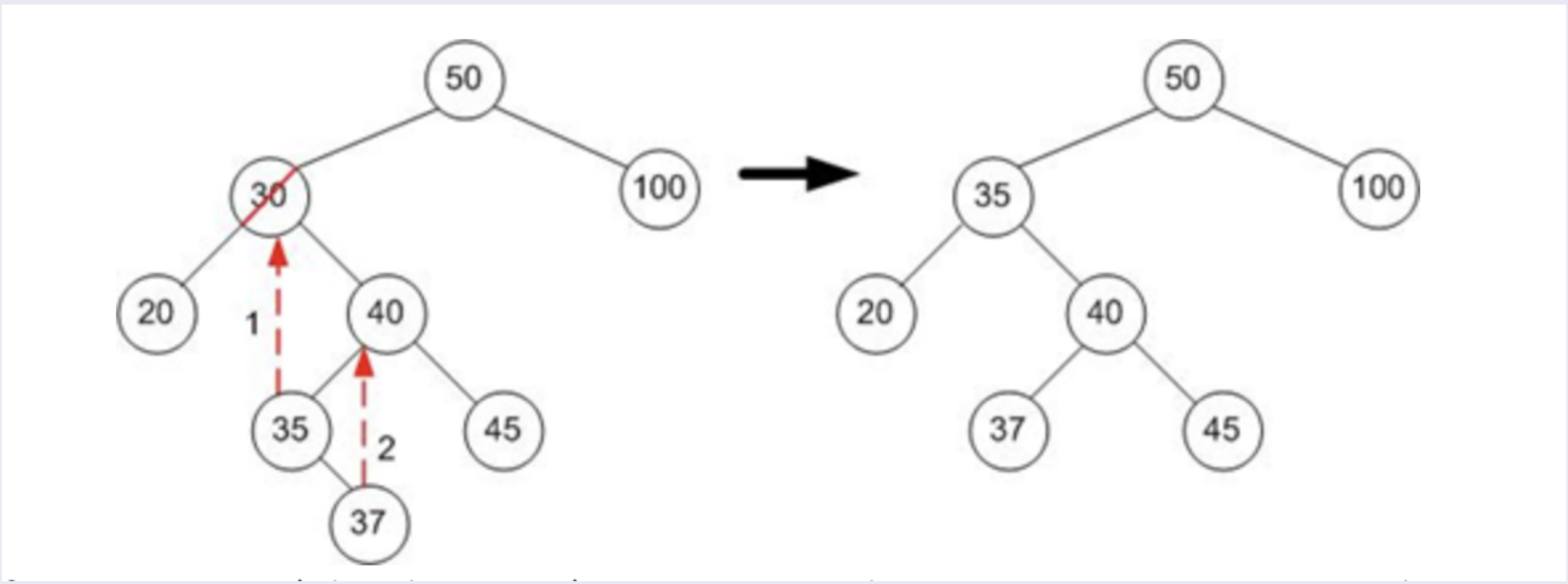
Árvores Binárias

Removendo um nó que possui DOIS filhos:



Árvores Binárias

Removendo um nó que possui DOIS filhos: Pode-se escolher como substituto um filho que tenha outros filhos, e esses filhos precisam ser realocados também.



Remove Nó da Árvore Binária

```
NO *removeDaArvore(NO *nodo, int chave){
    NO *aux, *aux1;
    if (nodo == NULL) printf("Árvore vazia!");
    else if(nodo->chave > chave) //Procurando a chave para remover
        nodo->filhoEsq = removeDaArvore(nodo->filhoEsq, chave);
    else if (nodo->chave < chave) //Procurando a chave para remover
        nodo->filhoDir = removeDaArvore(nodo->filhoDir, chave);
    else if ((nodo->filhoEsq == NULL) && (nodo->filhoDir == NULL)){  //Achou a chave
        //Aqui temos um nó folha a ser removido
        free(nodo);
        nodo = NULL;
    }else if((nodo->filhoEsq != NULL) && (nodo->filhoDir != NULL)){
        //Aqui temos um nó com dois filhos a ser removido
        //Substituiremos sempre pelo filho da subárvore à direita
        aux = buscaMaisAEsquerda(nodo);
        aux1=buscaFilhoEsq(nodo,aux->chave);
        if (aux->filhoDir!=NULL)
           aux1->filhoEsq = aux->filhoDir;
        else aux1->filhoEsq = NULL;
        nodo->chave = aux->chave;
    }else if(nodo->filhoEsq == NULL){
        //Aqui temos um nó que tem apenas o filho da direita a ser removido
        nodo = nodo->filhoDir;
    }else if(nodo->filhoDir == NULL){
        //Aqui temos um nó que tem apenas o filho da esquerda a ser removido
        nodo = nodo->filhoEsq;
    }
    return nodo;
}
```

Busca Nó mais à Esquerda da Subárvore da Direita

```
NO *buscaMaisADireita(NO *nodo){
    NO *aux;
    for(aux = nodo; aux->filhoDir!=NULL;aux=aux->filhoDir);
    return aux;
}

NO *buscaMaisAEsquerda(NO *nodo){
    NO *aux;
    for(aux=nodo; aux->filhoEsq!=NULL; aux=aux->filhoEsq);
    return aux;
}
```

Caso do Nó mais à Esquerda da Subárvore da Direita possua filhos, realoca eles. Para tanto, precisa localizar o PAI do Nó mais à Esquerda da Subárvore à Direita, pois ele passará a ser o pai dos filhos do Nó Substituto do removido.

```
NO *buscaFilhoEsq(NO *nodo, int chave){
  if (nodo->filhoEsq == NULL)
      return NULL;
  else if(nodo->filhoEsq->chave != chave)
      return buscaFilhoEsq(nodo->filhoEsq, chave);
  else return nodo;
}
```

Percursos

Árvores Binárias

Percursos:

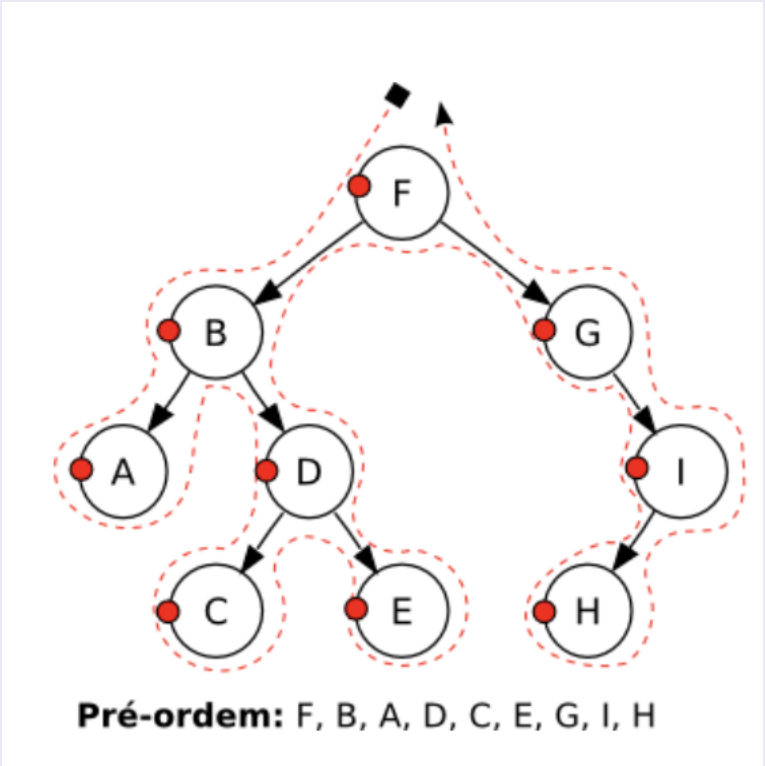
Existe 3 formas possíveis de se percorrer uma Árvore Binária:

- Pré-Ordem: RED = raiz, esquerda, direita
- Em-Ordem: ERD = esquerda, raiz, direita
- Pós-Ordem: EDR = esquerda, direita, raiz

Árvores Binárias

Percurso Pré-Ordem:

Percorre a Árvore na ordem RED → raiz, esquerda, direita



Árvores Binárias

Percurso Pré-Ordem:

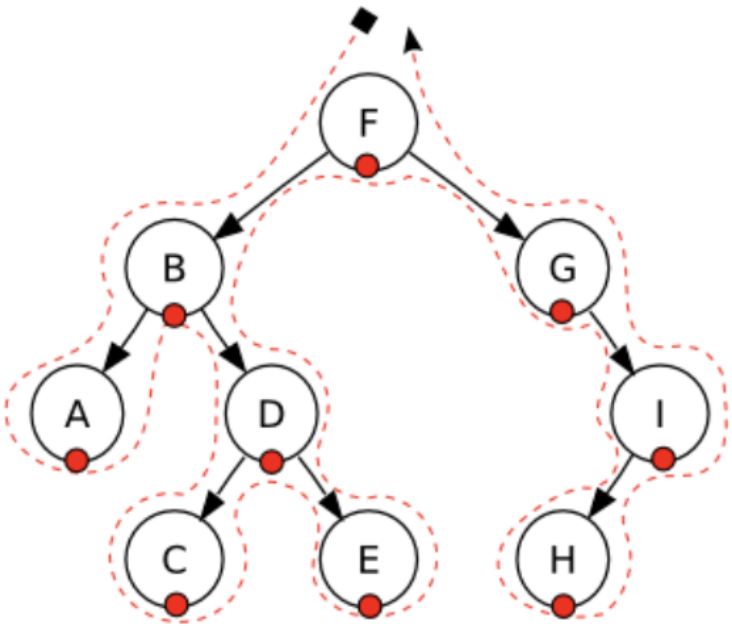
Percorre a Árvore na ordem RED → raiz, esquerda, direita

```
void preOrdem(NO *nodo){
  if(nodo){
    printf("%d\n",nodo->chave);
    preOrdem(nodo->filhoEsq);
    preOrdem(nodo->filhoDir);
  }
}
```

Árvores Binárias

Percurso Em-Ordem:

Percorre a Árvore na ordem ERD → esquerda, raiz, direita



Ordem simétrica: A, B, C, D, E, F, G, H, I

Árvores Binárias

Percurso Em-Ordem:

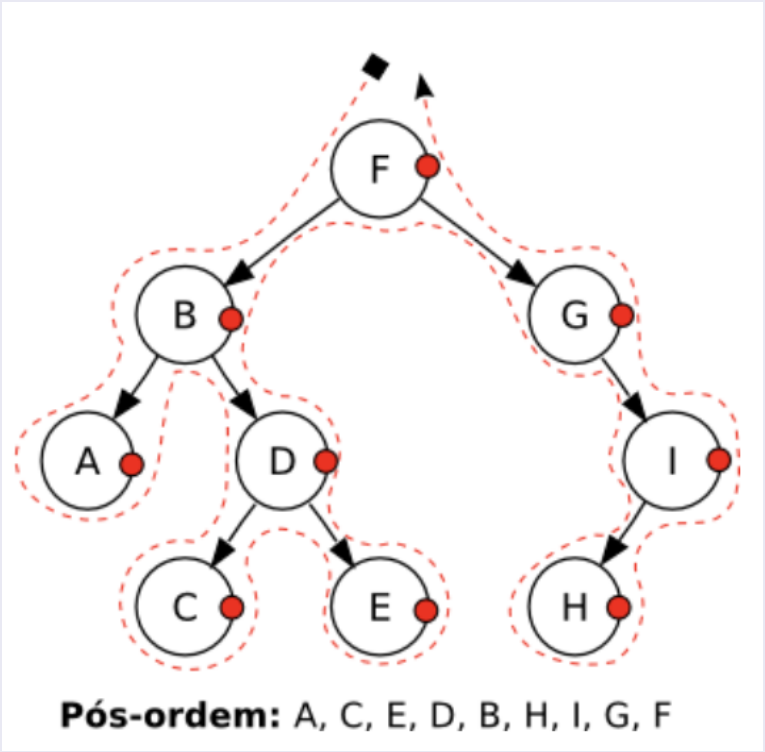
Percorre a Árvore na ordem ERD → esquerda, raiz, direita

```
void emOrdem(NO *nodo){
   if(nodo){
      emOrdem(nodo->filhoEsq);
      printf("%d\n",nodo->chave);
      emOrdem(nodo->filhoDir);
   }
}
```

Árvores Binárias

Percurso Pós-Ordem:

Percorre a Árvore na ordem EDR → esquerda, direita, raiz



Árvores Binárias

Percurso Pós-Ordem:

Percorre a Árvore na ordem EDR → esquerda, direita, raiz

```
void posOrdem(NO *nodo){
  if(nodo){
    posOrdem(nodo->filhoEsq);
    posOrdem(nodo->filhoDir);
    printf("%d\n",nodo->chave);
  }
}
```

Árvores Binárias

Desenhar a Árvore:

Vamos desenhar a Árvore da forma mais simples, mostrando a árvore deitada, primeira a raiz, depois seus filhos, nível a nível.

```
void imprimir(NO *nodo, int nivel){
  int i;
  if(nodo != NULL) {
    imprimir(nodo->filhoDir, nivel + 1);
    for(i = 0; i <= nivel-1; i++) printf("   ");
    printf("+--");
    printf("%d\n",nodo->chave);
    imprimir(nodo->filhoEsq, nivel + 1);
  }
  return;
}
```

MENU

```c
int menu(void){
  int opcao;
  printf("Opções:\n");
  printf("-------\n");
  printf("0. Insere elementos na Árvore Binária Ordenada\n");
  printf("1. Mostra Árvore Binária Ordenada em Pré-ordem\n");
  printf("2. Mostra Árvore Binária Ordenada em Pós-ordem\n");
  printf("3. Mostra Árvore Binária Ordenada Em Ordem\n");
  printf("4. Desenha a Árvore Binária Ordenada da raiz às folhas\n");
  printf("5. Remove elementos da Árvore Binária Ordenada\n");
  printf("6. Busca elementos na Árvore Binária Ordenada\n");
  printf("7. Fechar\n\n");

  do{
    printf("Escolha [0,1,2,3,4,5,6 ou 7]: ");
    scanf("%d",&opcao);
    printf("\n\n");
  }
  while((opcao < 0) && (opcao > 7));
  return opcao;
}
```

MAIN

```
void main() {
  NO *raiz = NULL, *nodo;
  int escolha, chave;
  do{
    escolha = menu();
    switch(escolha){
        case 0:
            printf("Insere elementos na Árvore Binária Ordenada \n\r");
            while(escolha == 0){
                printf("Digite o elemento que deseja inserir ou -1 para sair: ");
                scanf("%d",&chave);
                if (chave != -1)
                    raiz = insereNaArvore(raiz,chave);
                else break;
            }
            break;
        case 1:
            printf("Percorre a Árvore Binária Ordenada e mostra no modo Pré-ordem\n\r");
            preOrdem(raiz);
            break;
        case 2:
            printf("Percorre a Árvore Binária Ordenada e mostra no modo Pós-ordem\n\r");
            posOrdem(raiz);
            break;
        case 3:
            printf("Percorre a Árvore Binária Ordenada e mostra no modo Em Ordem\n\r");
            emOrdem(raiz);
            break;
```

MAIN

```
        case 4:
            printf("Mostra Árvore Binária Ordenada (raiz à esquerda e folhas à direita)\n\r");
            imprimir(raiz, 0);
            break;
        case 5:
            printf("Remove elementos da Árvore Binária Ordenada \n\r");
            while(escolha == 5){
                printf("Digite o elemento que deseja remover ou -1 para sair: ");
                scanf("%d",&chave);
                if (chave != -1)
                    raiz = removeDaArvore(raiz,chave);
                else break;
            }
            break;
        case 6:
            printf("Busca elemento na Árvore Binária Ordenada \n\r");
            while(escolha == 6){
                printf("Digite o elemento que deseja buscar ou -1 para sair: ");
                scanf("%d",&chave);
                if (chave != -1)
                    nodo = buscaNaArvore(raiz,chave);
                else break;
            }
            break;
        }
        printf("\n\nDigite qualquer tecla...");
        getchar();
    }
while(escolha != 7);
destroi(raiz);
```

Referências

Referências


CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.